Computational Complexity Theory

Lecture 18: P-completeness; Parity not in AC^0

Department of Computer Science,
Indian Institute of Science

Recap: Karp-Lipton theorem

- Theorem (*Karp & Lipton 1982*). If NP $\subsetneq$ P/poly then PH = Σ_2 .

- If we can show NP $\not\subset$ P/poly assuming P $\neq$ NP , then

  NP $\not\subset$ P/poly $\Leftrightarrow$ P $\neq$ NP .

- Karp-Lipton theorem shows NP $\not\subset$ P/poly assuming the stronger statement PH $\neq$ Σ_2 .

Recap: Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!
- Theorem. *(Iwama, Lachish, Morizumi & Raz 2002)* There is a language $L \in NP$ such that any circuit C_n that decides $L \cap \{0,1\}^n$ requires $5n - o(n)$ many $\wedge$ and $\vee$ gates.

Recap: Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Fact. PARITY(x_1, x_2, …, x_n) can be computed by a circuit of size $O(n)$ and a formula of size $O(n^2)$.
- Theorem. *(Khrapchenko 1971)* Any formula computing PARITY(x_1, x_2, …, x_n) has size $\Omega(n^2)$.

Recap: Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Theorem. (*Andreev 1987, Hastad 1998*) There's a f that can be computed by a O(n)-size circuit such that any formula computing f has size $\Omega(n^{3-o(1)})$.

Recap: Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Conjecture. *(Circuits more powerful than formulas)* There's a f that can be computed by a O(n)-size circuit such that any formula computing f has size $n^{\omega(1)}$.

Recap: Non-uniform size hierarchy

- Shanon's result. There's a constant $c \geq 1$ such that every Boolean function in n variables has a circuit of size at most $c.(2^n/n)$.

- Theorem. There's a constant $d \geq 1$ s.t. if $T_1: N \rightarrow N$ & $T_2: N \rightarrow N$ and $T_1(n) \leq d^{-1}.T_2(n) \leq T_2(n) \leq c.(2^n/n)$ then

$$\mathrm{SIZE}(T_1(n)) \subsetneq \mathrm{SIZE}(T_2(n)).$$

Recap: Class NC

- NC stands for Nick's Class – named after Nick Pippenger.
- Definition. For $i \in \mathbb{N}$, a language L is in NC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $NC = \bigcup_{i \in \mathbb{N}} NC^i$.

- PARITY is in NC^1 = *poly(n)*-size Boolean formulas.

Recap: Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$. (stands for *Alternating Class*)

- Observation. $AC^i \subseteq NC^{i+1} \subseteq AC^{i+1}$ for all $i \geq 0$.

Replace an unbounded fan-in gate by a binary tree of bounded fan-in gates.

Recap: Class AC

- Definition. For $i \in \mathbb{N} \cup \{0\}$, a language L is in AC^i if there is a polynomial function $q(.)$ and a constant c s.t. L is decided by a $q(n)$-size **unbounded fan-in** circuit family $\{C_n\}_{n \in \mathbb{N}}$, where depth of C_n is at most $c.(\log n)^i$ for every $n \in \mathbb{N}$.

- Definition. $AC = \bigcup_{i \geq 0} AC^i$.

- In this lecture, we'll show that PARITY is not in AC^0, i.e., $AC^0 \subsetneq NC^1$.

P-completeness

P-completeness

- Recall, to define completeness of a complexity class, we need an appropriate notion of a *reduction*.
- What kind of reductions will be suitable is guided by *a complexity question*, like a comparison between the complexity class under consideration & another class.
- Is P = (uniform) NC? Is P = L?…use log-space reduction!

- Definition. A language $L \in P$ is *P-complete* if for every L' in P, $L' \leq_l L$.

P-complete problems

- Circuit value problem. Given a circuit and an input, compute the output of the circuit. (The reduction in the Cook-Levin theorem can be made a log-space reduction.)

- Linear programming. Check the feasibility of a system of linear inequality constraints over rationals. (Assignment problem)

- CFG membership. Given a context-free grammar and a string, decide if the string can be generated by the grammar.

No log-space algo for PC problems

- Theorem. Let L be a P-complete language. Then,

  L is in L ⟺ P = L.

- Proof. Easy.

- Can't hope to get a log-space algorithm for a P-complete problem unless P = L.

No parallel algo for PC problems

- Theorem. Let L be a P-complete language. Then,

  L is in NC ⟺ $P \subseteq NC$.

- Proof. ⟸ direction is straightforward.

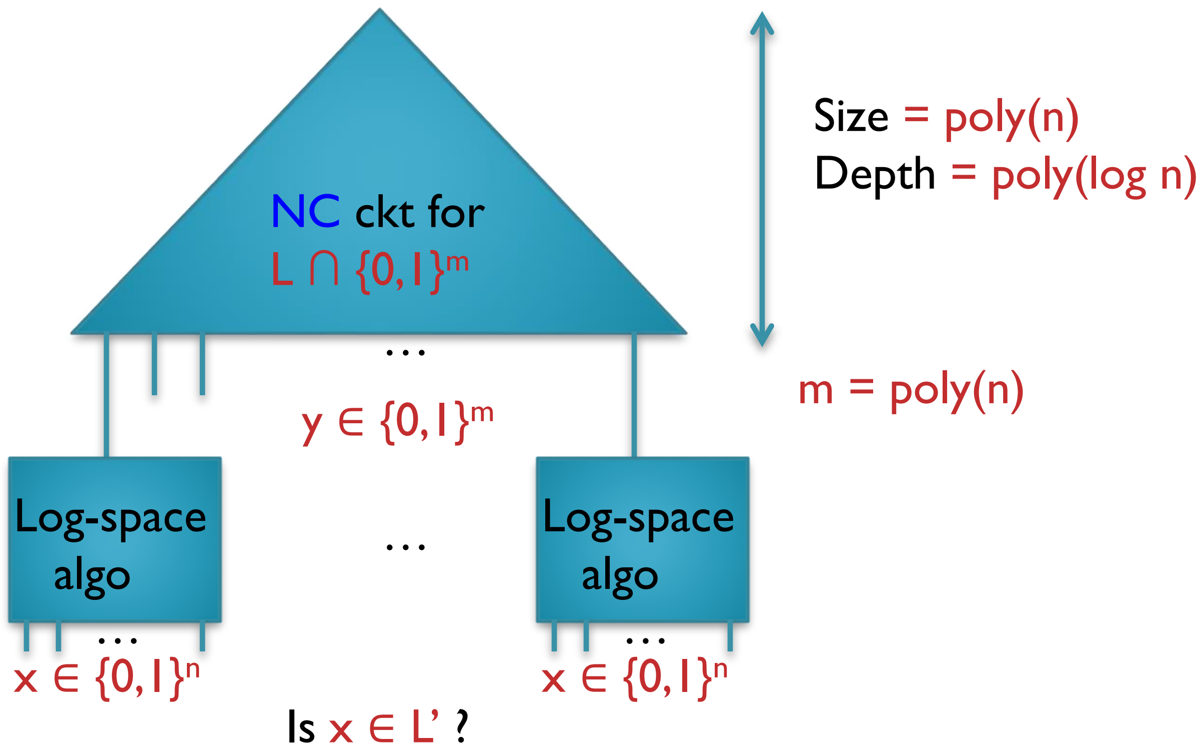
- Can't hope to get an efficient parallel algorithm for a P-complete problem unless $P \subseteq NC$.

No parallel algo for PC problems

- Theorem. Let L be a P-complete language. Then,

  L is in NC $\Leftrightarrow$ $P \subseteq NC$.

- Proof.($\Rightarrow$) Let $L' \in P$. As L is P-complete, $L' \leq_l L$.

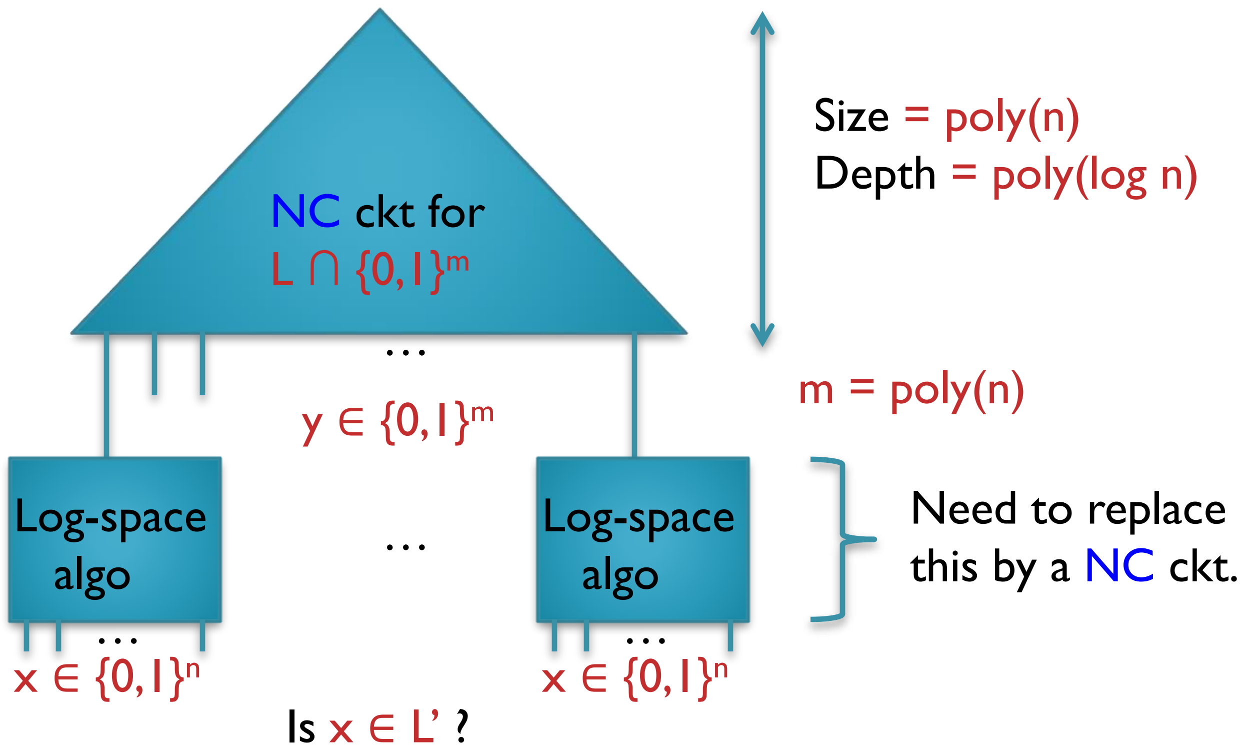


No parallel algo for PC problems

- Theorem. Let L be a P-complete language. Then,

  L is in NC $\Leftrightarrow$ $P \subseteq NC$.

- Proof.($\Rightarrow$) Let $L' \in P$. As L is P-complete, $L' \leq_l L$.



Parallelization of Log-space

- Do problems in L have efficient parallel algorithms? Yes!

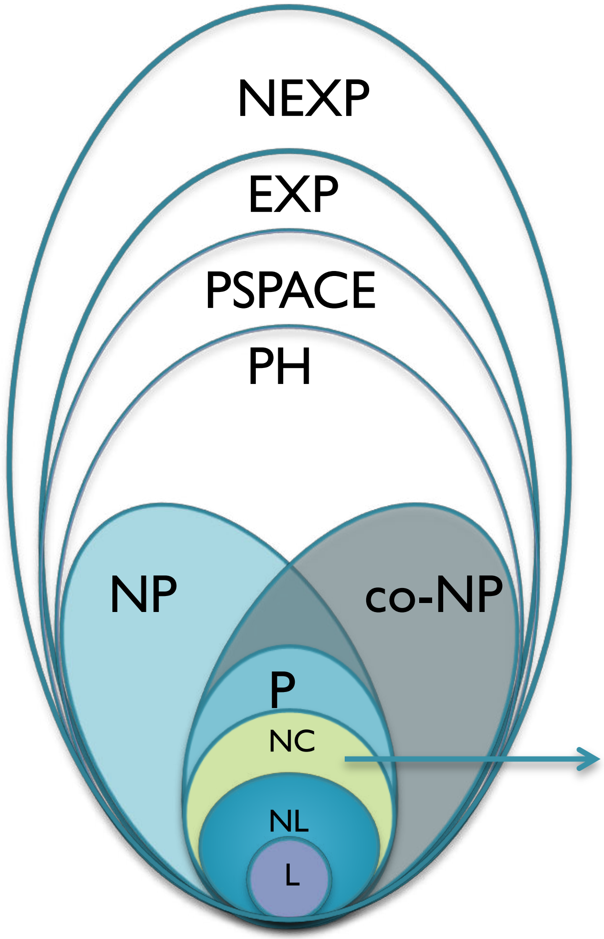
- Theorem. NL $\subseteq$ (uniform) NC. *(Assignment problem)*

Parallelization of Log-space

- Do problems in L have efficient parallel algorithms? Yes!

- Theorem. NL $\subseteq$ (uniform) NC. *(Assignment problem)*
- Proof sketch.
- 1. Construct the adjacency matrix A of the configuration graph.
- 2. Use repeated squaring of A to find out if there's a path from start to accept configurations.

Complexity zoo



(uniform) NC

In fact, (uniform) $NC^1 \subseteq L$ and $NL \subseteq$ (uniform) NC^2.

(Assignment)

The Parity function

The Parity function

- $\text{PARITY}(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$.

- Fact. $\text{PARITY}(x_1, x_2, \ldots, x_n)$ can be computed by a circuit of size $O(n)$ and a formula of size $O(n^2)$.

  circuit: has depth $O(\log n)$ — formula: has depth $O(\log n)$

- Theorem. *(Khrapchenko 1971)* Any formula computing $\text{PARITY}(x_1, x_2, \ldots, x_n)$ has size $\Omega(n^2)$.

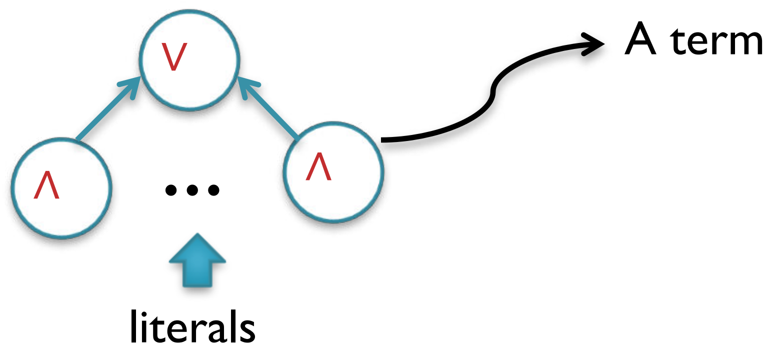
The Parity function

- $PARITY(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$.
- Fact. $PARITY(x_1, x_2, \ldots, x_n)$ can be computed by a circuit of size $O(n)$ and a formula of size $O(n^2)$.
- Theorem. *(Khrapchenko 1971)* Any formula computing $PARITY(x_1, x_2, \ldots, x_n)$ has size $\Omega(n^2)$.
- Can poly-size <u>constant depth</u> circuits compute PARITY? No!

Depth 2 circuit for Parity

- Without loss of generality, a depth 2 circuit is either a DNF or a CNF.

DNF:



- Any Boolean function can be computed by a DNF (similarly, CNF) with 2^n terms (respectively, clauses).

- Can we do better for depth 2 circuits computing PARITY?

Depth 2 circuit for Parity

- Without loss of generality, a depth 2 circuit is either a DNF or a CNF.

- Obs. Any DNF computing PARITY has ≥ 2^{n-1} terms.
- Proof. Let ϕ be a DNF computing PARITY. Then, every term in ϕ has n literals (otherwise, the value of PARITY can be fixed by fixing less than n variables which is false).

Depth 2 circuit for Parity

- Without loss of generality, a depth 2 circuit is either a DNF or a CNF.

- Obs. Any DNF computing PARITY has $\geq 2^{n-1}$ terms.
- Proof. Let ϕ be a DNF computing PARITY. Then, every term in ϕ has n literals (otherwise, the value of PARITY can be fixed by fixing less than n variables which is false). Such a term corresponds to a *unique* assignment that makes the term evaluate to 1. Terms corresponding to assignments that <u>set odd number of variables to 1</u> must be present in ϕ.

Depth 3 circuit for Parity

- Obs. There's a $2^{O(\sqrt{n})}$ size depth 3 circuit for PARITY.
- Proof.

$$\text{PARITY} = \underbrace{x_1 \oplus x_2 \oplus \dots \oplus x_{\sqrt{n}}}_{y_1} \oplus \dots \oplus \underbrace{x_{n-\sqrt{n}} \oplus x_2 \oplus \dots \oplus x_n}_{y_{\sqrt{n}}}$$

- *Divide & conquer*: Compute y_i and $\neg y_i$ by $2^{O(\sqrt{n})}$ size DNFs on the **x** literals. Compute $y_1 \oplus \dots \oplus y_{\sqrt{n}}$ by a $2^{O(\sqrt{n})}$ size CNF on the **y** literals. "Attach" the CNF with the DNFs and "merge" the two middle layers of $\vee$ gates.

Depth 3 circuit for Parity

- Obs. There's a $2^{O(\sqrt{n})}$ size depth 3 circuit for PARITY.
- Proof. $x_1 \oplus x_2 \oplus \dots \oplus x_{\sqrt{n}} \oplus \dots \oplus x_{n-\sqrt{n}} \oplus x_2 \oplus \dots \oplus x_n$

$$\text{PARITY} = \underbrace{x_1 \oplus x_2 \oplus \dots \oplus x_{\sqrt{n}}}_{y_1} \oplus \dots \oplus \underbrace{x_{n-\sqrt{n}} \oplus x_2 \oplus \dots \oplus x_n}_{y_{\sqrt{n}}}$$

PARITY = $y_1 \oplus \dots \oplus y_{\sqrt{n}}$

- *Divide & conquer*: Compute y_i and $\neg y_i$ by $2^{O(\sqrt{n})}$ size DNFs on the **x** literals. Compute $y_1 \oplus \dots \oplus y_{\sqrt{n}}$ by a $2^{O(\sqrt{n})}$ size CNF on the **y** literals. "Attach" the CNF with the DNFs and "merge" the two middle layers of $\vee$ gates.

Is the $2^{O(\sqrt{n})}$ upper bound on the size of depth 3 circuits computing PARITY tight? "Yes"

Depth d circuit for Parity

- Obs. There's a $\exp(n^{1/(d-1)})$ size depth d circuit for PARITY, where $\exp(x) = 2^x$. *(Homework)*
- Proof sketch. "Divide & conquer" for d-1 levels. Alternate between CNFs and DNFs. "Attach" the CNFs and the DNFs appropriately, and then "merge" the intermediate layers to bring the depth down to d.

- Is the $\exp(n^{1/(d-1)})$ upper bound on the size of depth d circuits computing PARITY tight? "Yes"

Lower bound for depth d circuits

- Theorem. *(Furst, Saxe, Sipser '81; Ajtai '83; Hastad '86)* Any depth d circuit computing PARITY has size $\exp(\Omega_d(n^{1/(d-1)}))$, where $\Omega_d()$ is hiding a d^{-1} factor.

- Furst, Saxe and Sipser showed a quasi-polynomial lower bound.
- Ajtai showed an exponential lower bound, but the bound wasn't optimal.
- Hastad showed an $\exp(\Omega(n^{1/(d-1)}))$ lower bound.
- *Rossman (2015)* showed an optimal $\exp(\Omega(dn^{1/(d-1)}))$ lower bound.

Lower bound for depth d circuits

- Theorem. *(Furst, Saxe, Sipser '81; Ajtai '83; Hastad '86)* Any depth d circuit computing PARITY has size $\exp(\Omega_d(n^{1/(d-1)}))$, where $\Omega_d()$ is hiding a d^{-1} factor.

- Gives a super-polynomial lower bound for depth d circuits for d up to o(log n).

- A lower bound for circuits of depth d = O(log n) implies a Boolean formula lower bound!

Lower bound for depth d circuits

- Theorem. *(Furst, Saxe, Sipser '81; Ajtai '83; Hastad '86)* Any depth d circuit computing PARITY has size $\exp(\Omega_d(n^{1/(d-1)}))$, where $\Omega_d()$ is hiding a d^{-1} factor.
- Proof idea. A ***random assignment*** to a "large" fraction of the variables makes a constant depth circuit of polynomial size evaluate to a constant (i.e., the circuit stops depending on the unset variables). On the other hand, we cannot make PARITY evaluate to a constant by setting less than n variables.

Lower bound for depth d circuits

- Theorem. *(Furst, Saxe, Sipser '81; Ajtai '83; Hastad '86)* Any depth d circuit computing PARITY has size $\exp(\Omega_d(n^{1/(d-1)}))$, where $\Omega_d()$ is hiding a d^{-1} factor.
- Proof idea. A ***random assignment*** to a "large" fraction of the variables makes a constant depth circuit of polynomial size evaluate to a constant (i.e., the circuit stops depending on the unset variables).

↓

- We'll prove this fact using Hastad's **Switching lemma**. But first let us discuss some structural simplifications of depth d circuits.

Lower bound for depth d circuits

- Theorem. *(Furst, Saxe, Sipser '81; Ajtai '83; Hastad '86)* Any depth d circuit computing PARITY has size $\exp(\Omega_d(n^{1/(d-1)}))$, where $\Omega_d()$ is hiding a d^{-1} factor.
- Proof idea. A ***random assignment*** to a "large" fraction of the variables makes a constant depth circuit of polynomial size evaluate to a constant (i.e., the circuit stops depending on the unset variables).

↓

- We'll prove this fact using Hastad's **Switching lemma**. But first let us discuss some structural simplifications of depth d circuits.

Will be proved in the next lecture

Simplifying depth d circuits

- Fact 1. If $f(x_1,\ldots, x_n)$ is computable by a circuit of depth d and size s, then f is also computable by a circuit C of depth d and size $O(s)$ such that C has <u>no ¬ gates</u> and the inputs to C are $x_1, \ldots, x_n$ and $\neg x_1, \ldots, \neg x_n$.

Simplifying depth d circuits

- Fact 1. If $f(x_1,\dots, x_n)$ is computable by a circuit of depth d and size s, then f is also computable by a circuit C of depth d and size $O(s)$ such that C has no $\neg$ gates and the inputs to C are $x_1, \dots, x_n$ and $\neg x_1, \dots, \neg x_n$.
- Fact 2. If f is computable by a circuit of depth d and size s, then f is also computable by a *formula* of depth d and size $O(s)^d$.

Simplifying depth d circuits

- Fact 1. If $f(x_1,\ldots,x_n)$ is computable by a circuit of depth d and size s, then f is also computable by a circuit C of depth d and size $O(s)$ such that C has no ¬ gates and the inputs to C are $x_1, \ldots, x_n$ and $\neg x_1, \ldots, \neg x_n$.
- Fact 2. If f is computable by a circuit of depth d and size s, then f is also computable by a *formula* of depth d and size $O(s)^d$.
- Fact 3. If f is computable by a formula of depth d and size s, then f is computable by a formula C of depth d and size $O(sd)$ that has alternating layers of $\vee$ and $\wedge$ gates with inputs feeding into *only* the bottom layer.

Simplifying depth d circuits

- Fact 1. If $f(x_1,\ldots, x_n)$ is computable by a circuit of depth d and size s, then f is also computable by a circuit C of depth d and size $O(s)$ such that C has no $\neg$ gates and the inputs to C are $x_1, \ldots, x_n$ and $\neg x_1, \ldots, \neg x_n$.
- Fact 2. If f is computable by a circuit of depth d and size s, then f is also computable by a *formula* of depth d and size $O(s)^d$.
- Fact 3. If f is computable by a formula of depth d and size s, then f is computable by a formula C of depth d and size $O(sd)$ that has alternating layers of $\vee$ and $\wedge$ gates with inputs feeding into *only* the bottom layer.

Homework: Prove the above facts.

Random restrictions

- A *restriction* σ is a partial assignment to a subset of the n variables.

- A *random restriction* σ that leaves m variables alive/unset is obtained by picking a random subset $S \subseteq [n]$ of size $n\text{-}m$ and setting every variable in S to $0/1$ uniformly and independently.

- Let f_σ denote the function obtained by applying the restriction σ on f.

The Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < ½$. Then,

  $\Pr_\sigma$ [f_σ can't be represented as a k-DNF] $\leq (16pt)^k$.